:521-61 / 99 ... N95- 19768

p-13

DYNACLIPS (DYNAmic CLIPS): A DYNAMIC KNOWLEDGE EXCHANGE TOOL FOR INTELLIGENT AGENTS

Authors

Yilmaz Cengeloglu
PO Box 4142, Winter Park, FL, 32793-4142
E-mail: yil@engr.ucf.edu

Soheil Khajenoori, Assoc. Prof., Embry-Riddle Aeronautical University,
Department of Computer Science,
Daytona Beach, FL, 32114, E-mail: soheil@erau.db.erau.edu

Darrell Linton, Assoc. Prof., University of Central Florida,
Department of Electrical and Computer Engineering,
Orlando, FL, 32816, E-mail: dgl@engr.ucf.edu

ABSTRACT

In a dynamic environment, intelligent agents must be responsive to unanticipated conditions. When such conditions occur, an intelligent agent may have to stop a previously planned and scheduled course of actions and replan, reschedule, start new activities and initiate a new problem solving process to successfully respond to the new conditions. Problems occur when an intelligent agent does not have enough knowledge to properly respond to the new situation. DYNACLIPS is an implementation of a framework for dynamic knowledge exchange among intelligent agents. Each intelligent agent is a CLIPS shell and runs a separate process under SunOS operating system. Intelligent agents can exchange facts, rules, and CLIPS commands at run time. Knowledge exchange among intelligent agents at run time does not effect execution of either sender and receiver intelligent agent. Intelligent agents can keep the knowledge temporarily or permanently. In other words, knowledge exchange among intelligent agents would allow for a form of learning to be accomplished.

1. INTRODUCTION

Applications of expert systems to variety of problems are growing rapidly. As the size and complexity of these systems grow, integration of independent cooperating expert systems is becoming a potential solution approach to large scale applications. In this paper, the blackboard model of distributed problem solving is discussed and architecture, implementation and usage of DYNACLIPS is explained.

1.1. Distributed Problem Solving (DPS)

Distributed problem solving in artificial intelligence is a research area which deals with solving a problem in a distributed environment through planning and cooperation among a set of intelligent entities *(i.e., agents)*. Each intelligent agent can run in parallel with other intelligent agents. Intelligent agents may be geographically distributed or operate within a single computer. An intelligent agent may possess simple processing elements or a complex rational behavior. A paramount issue in DPS is the communication and information sharing among participating intelligent agents, necessary to produce a solution. The blackboard model of problem solving is one of the most common approaches in the distributed artificial intelligence area. In the following section we will focus on blackboard architecture as a model for distributed problem solving.

1.2. Blackboard Model of Distributed Problem Solving

The blackboard architecture (BBA) is one of the most inspiring cooperative problem solving paradigms in artificial intelligence. The approach had generated much interest among researchers. BBA is a relatively complex problem solving model prescribing the organization of knowledge, data and the problem-solving behavior within an overall organization. The blackboard model is becoming a useful tool for complex applications whose solution requires a set of separate though interrelated sources of knowledge and expertise.

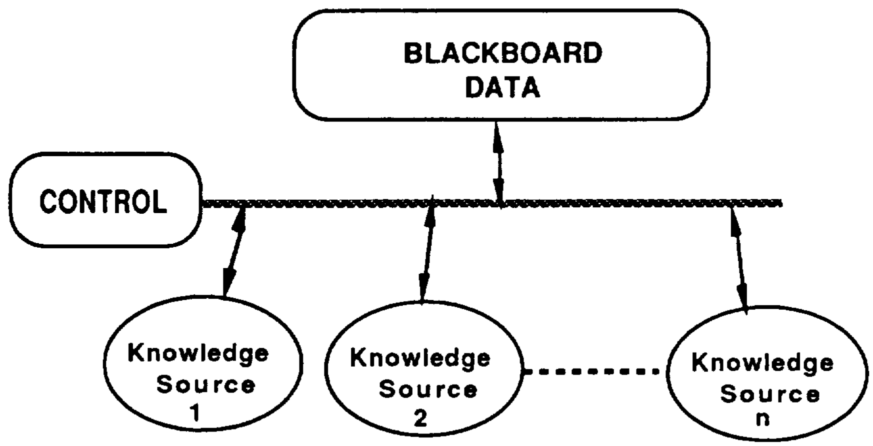

Figure 1. Blackboard Model For DSP

A blackboard model contains blackboard, control, and knowledge sources. Figure 1 shows a basic blackboard model for DPS applications. The Knowledge Sources are the knowledge needed to solve the problem; they are kept separate and independent. Each knowledge source can use different knowledge representations techniques. The Blackboard Data is a global database that contains problem-solving states. The knowledge sources produce changes to the blackboard that lead incrementally to the solution of the problem being solved. Communication and interaction among the knowledge sources takes place solely through the blackboard. The Control determines the area of the problem solving space on which to focus. The focus of attention can be either the knowledge sources, the blackboard or a combination of both. The solution is built one step at a time by using a cooperation of different knowledge sources. The blackboard model is a complete parallel and distributed computation model. The parallel blackboard model involves the parallel execution of knowledge sources and the control component. The distributed blackboard model involves the communication of blackboard data among blackboard subsystems. The main issue here is to decide what to communicate and where and when to send data. The blackboard systems are being used increasingly in real time systems. Architectural extensions to the basic blackboard model can be added to increase its performance for real time applications. [1,2,3]

1.3. An Overview of Existing BBA Tools

A number of BBA tools are reported in the literature. Here we provide an overview of some of these systems.

BB_CLIPS (Blackboard CLIPS) [4, 5] is an extended version of CLIPS version 4.3 developed by National Research Council of Canada. In BB_CLIPS, each CLIPS rule or group of rules serves as a knowledge source. The fact base of BB_CLIPS serves as the blackboard, and its agenda manager serves as the scheduler.

RT-1 architecture [6] is a small-scale, coarse-grained, distributed architecture based on the blackboard model. It consists of a set of reasoning modules which share a common blackboard data and communicate with each other by "signaling events".

PRAIS (Parallel Real-Time Artificial Intelligence Systems) [7] is an architecture for real-time artificial intelligence system. It provides coarse-grained parallel execution based upon a virtual global memory. PRAIS has operating system extensions for fact handling and message passing among multiple copies of CLIPS.

MARBLE (Multiple Accessed Rete blackboard linked Experts) [8] is a system that provides parallel environment for cooperating expert systems. Blackboard contains facts related to the problem being solved and it is used for communication among expert systems. Each expert shell in the system keeps a copy of the blackboard in its own fact base. Marble has been used to implement a multi-person blackjack simulation.

AI Bus [9] is a software architecture and toolkit that supports the construction of large-scale cooperating systems. An agent is the fundamental entity in the AI bus and communicates with other agents via message passing. An agent has goals, plans, abilities and needs that other agents used for cooperation.

GBB (Generic Blackboard) [10,11] is toolkit for developers needs to construct a high-performance blackboard based applications. The focus in GBB is increasing the efficiency of blackboard access, especially for pattern-based retrieval. GBB consist of different subsytems : a blackboard database development subsystem, control shells, knowledge source representation languages and graphic displays for monitoring and examining blackboard and control components. GBB is an extension of Common Lisp and CLOS (Common Lisp Object System).

GEST (Generic Expert System Tool) [12] has been developed by Georgia Tech Research Institute. The main components of the GEST are the central blackboard data structure, independent experts or knowledge sources and the control module. The blackboard data structure holds the current state of the problem solving process. It is also common communication pathway among knowledge sources.

CAGE and POLIGON [13] have been developed at Stanford University. They are two different frameworks for concurrent problem solving. CAGE is a conventional blackboard system which supports parallelism at knowledge sources level. The knowledge source, the rules in the knowledge source, or clauses in a rule can be executed in parallel. CAGE is a shared memory multiprocessing system. POLIGONs' functionality is similar to CAGE.

Hearsay-II [2,3,14] is a speech understanding system developed at Carnegie-Mellon University. Hearsay-II provides a framework that different knowledge sources cooperate to solve a problem.

More recently, significant work is being done to develop Knowledge Interchange Formats (KIF) and Knowledge Query and Manipulation languages (KQML) by Stanford University [15,16]. KIF is a computer-oriented language for the interchange of knowledge among disparate programs that is written by different programmers, at different times, in different languages. KIF is not a language for the internal representation of knowledge. When a program reads a knowledge base in KIF, it converts the knowledge into its own internal form. When the program needs to communicate with another program, it maps its internal data structures into KIF. KQML messages are similar to KIF expressions. Each Messages in KQML is one piece of a dialogue between the sender and receiver programs.

2. IMPLEMENTATION OF DYNACLIPS

Using the SunOS operating system multiprocessing techniques and interprocess communication facilities, we have developed a prototype system on a Sun platform to demonstrate the dynamic knowledge exchange among intelligent agents. In the following sections we provide a short overview of SunOS InterProcess Communication Facilities (IPC) and describe the implementation aspect of the DYNACLIPS.

2.1. InterProcess Communication Facilities (IPC)

Interprocess Communication involves sharing data between processes and coordinating access to shared data. The SunOS operating system provides several facilities and mechanism by which processes can communicate. These include Messages, Semaphores and Shared Memory.

The *Messaging facility* provides processes with a means to send and receive messages, and to queue messages for processing in an arbitrary order. Messages can be assigned specific types and each would have an explicit length. Among other uses, this allows a server process to direct message traffic between multiple clients on its queue. Messages sent to the queue can be of variable size. The application programmer must insure that the queue space limitations are not exhausted when more than one process uses the same queue. The process owning the queue must establish the read/write permissions to allow/deny other processes access to the queue. Furthermore, it is the responsibility of the owner process to remove the queue when it is no longer in use or prior to exiting.

Semaphores provide a mechanism by which processes can query or alter status information. They are often used to monitor and control the availability of system resources, such as shared memory segments. Semaphores may be operated as individual units or as elements in a set. A semaphore set consists of a control structure and array of individual semaphores.

Shared memory allows more than one process at a time to attach a segment of physical memory to its virtual address space. When write access is allowed for more than one process, an outside protocol or mechanism such as a semaphore can be used to prevent contentions [17].

2.2. Architecture of DYNACLIPS

In this section we provide an overview and discussion of each component of the DYNACLIPS. Figure 2 represents the overall architecture of the DYNACLIPS. Shared memory has been used to implement the system main blackboard to broadcast messages from control to intelligent agents. Message queues have been used to transfer messages from the intelligent agents to the control. Semaphores are used to make intelligent agents and control to sleep or wake up in order to reduce the load on CPU. Each intelligent agent and control has an input/output port to the world outside of the application framework to interface with the other processes outside of their environment. These outside processes might be any program that uses the application framework. In DYNACLIPS, intelligent agents and control can also use IPC facilities to interface with the outside programs.

2.3. Control

The control component of the system has been implemented using the C programming language. It runs as a separate process and communicates with the other processes using IPC facilities. The control can be loaded and executed by entering the word "control" at SunOS prompt. The control always has to be loaded first. Once the control is loaded, it creates the three incoming control message queues for the requests coming from the intelligent agents, one shared memory to be used as the main system blackboard and two semophores for control and intelligent agents. The control has read/write access to the main system blackboard and has only read access from the message queues. If there is no request from intelligent agents, control sleeps until any agent makes a request.

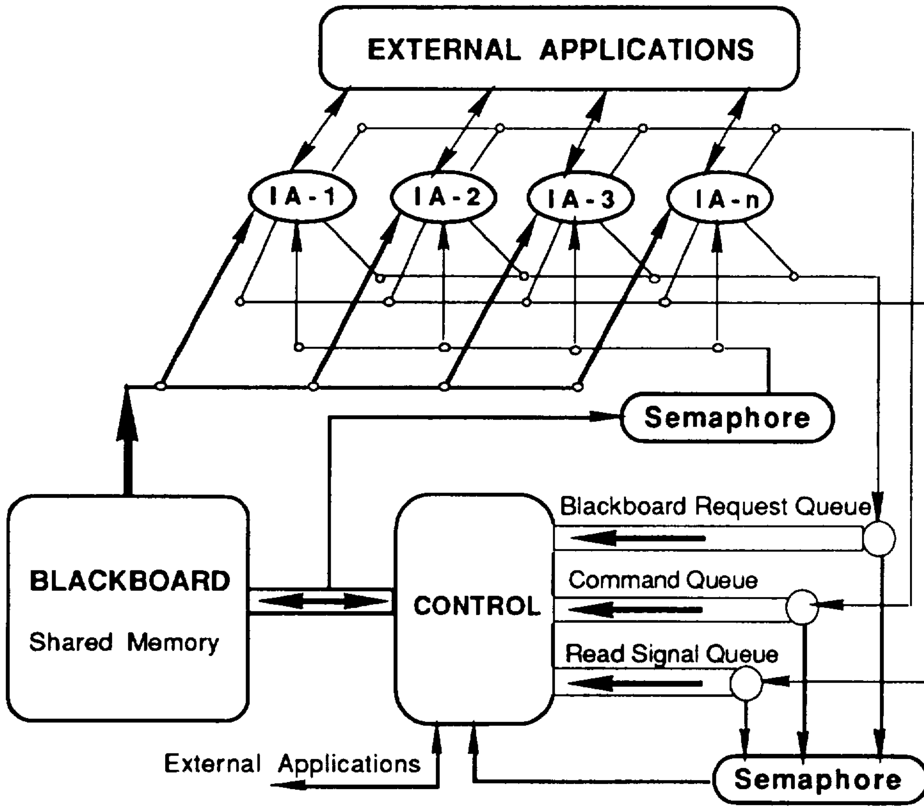

Figure 2. Architecture of DYNACLIPS

The DYNACLIPS takes advantage of the multiprocessing capabilities of the SunOS operating system. A message facility has been used for setting up a communication link from the intelligent agents to the control. The control creates incoming queues to receive messages from the intelligent agents. These control message queues use "First-In First-Out" (FIFO) methodology as shown in Figure 3.

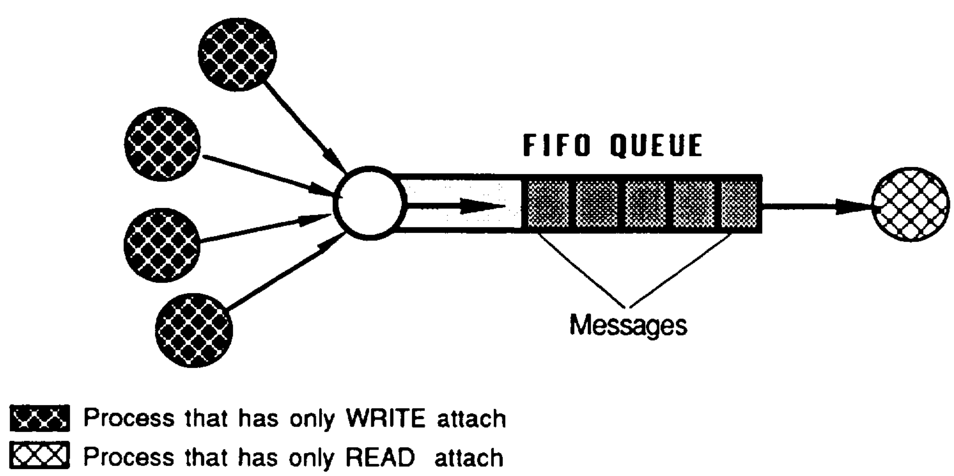

Figure 3. Message Transfer Among Processes

In order to communicate with the control process, intelligent agents send their requests to the control message queues. Intelligent agents have only write permission to these queues. Message queue facility of IPC can also be utilized to allow the control module and/or intelligent agents to communicate with other application programs running outside of their environment.

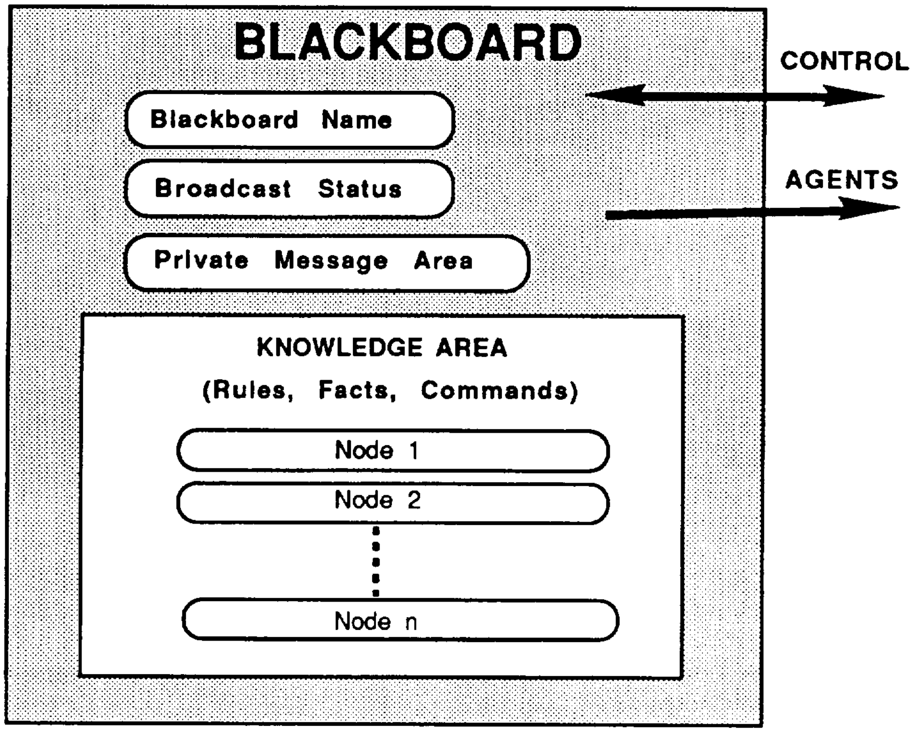

Figure 4. Blackboard

2.4. Blackboard

The blackboard is a shared memory that holds information and knowledge that intelligent agents use. The blackboard is organized by the control component. The blackboard has four different members as shown in Figure 4. These members are: 1) *Blackboard name* which holds the name of the blackboard that is being used, this is necessary if multiple blackboards are in use, 2) *Blackboard status* which is a counter that is incremented after each update of the blackboard, 3) *Private message area* which is used by the control component to send system related messages to the intelligent agents, 4) The *knowledge area* which holds, rules, facts and commands that needs to be broadcast to the intelligent agents.

Shared Memory facility of IPC has been used to broadcast messages to all intelligent agents via the control. Figure 5 represents broadcasting data using shared memory configuration. When a process that has a write attach, writes a message to the shared memory, this message will be visible to all processes that have read attach to shared memory. When there is more than one process able to write to shared memory, semaphores should be used to prevent processes accessing the same message space at the same time. In DYNACLIPS semaphores were not used for shared memory, because the control component is the only process that has read and write accesses to the shared memory and intelligent agents can only read from the shared memory.

In DYNACLIPS, there is a local blackboard for each intelligent agent and one main blackboard for the control. The DYNACLIPS uses the shared memory facility to implement the main system blackboard. This implementation was possible because all intelligent agents run on the same computer. If intelligent agents were distributed on different computer systems, then a copy of the main blackboard needs to be created and maintained in each computer system.

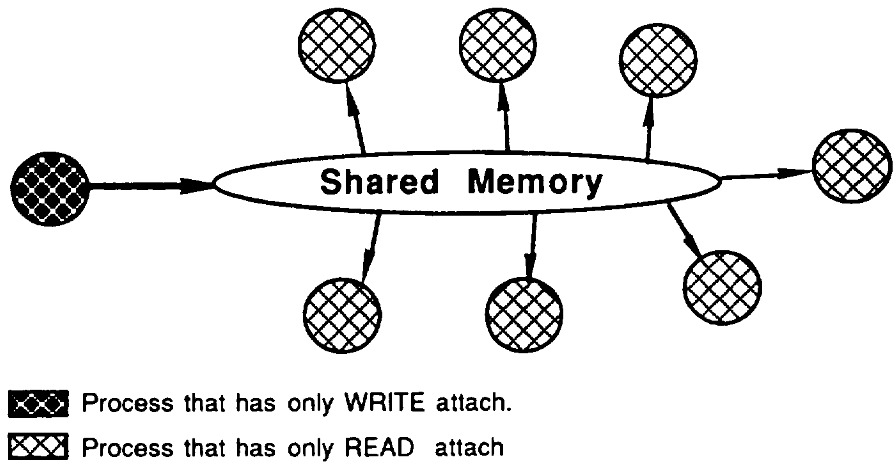

Figure 5. Message Broadcasting With Shared Memory

2.5. Intelligent Agents

An intelligent agent can be any application, such as an expert system shell, that is able to use IPC facilities on the SunOS operating system. In this prototype system, CLIPS (C Language Production System) was used as intelligent agents. CLIPS [18] is an expert system shell which uses a rule-based knowledge representation scheme. We have extended the CLIPS shell by adding a set of functions to provide the necessary capabilities to use IPC facilities.

The following command should be executed at SunOS prompt to load and activated a CLIPS based intelligent agent.

dynaclips <Knowledge base name > <Intelligent agent name> [Group name]

The *Knowledge base name* is a file that contains CLIPS rules. The *Intelligent agent name* and *Group name* are names given to intelligent agent by the user or application program that loads this knowledge base. Multiple copies of the CLIPS expert system shell, each representing an agent, can be loaded and activated at the same time using the above command. *Intelligent agent name* and *Group name* are also inserted as initial fact to intelligent agents. Following will be initial facts in the intelligent agents:

(knowledge name is *<Intelligent agent name>*)
(member of group *<Group name>*)

Once an intelligent agent is loaded, 1) it finds out the names of the control message queues and attaches itself to the queues as a writer. 2) It finds out the name of the main blackboard and attaches itself to it as a reader. 3) It sends a message to control component to inform that it has joined the system. If there is no rule to fire and nothing is changed in the blackboard, an intelligent agent sleeps until something changes in the blackboard.

As we mentioned previously, we have added number of functions to the CLIPS shell. The following functions can be used by an intelligent agent to communicate with the control as well as with other intelligent agents.

(ADD_BB_STR *<message string>)*
(DEL_BB_STR *<message string>)*
(CHANGE_GROUP_NAME *<new_group_name>)*
(EXIT_CLIPS)

ADD_BB_STR *(Add BlackBoard String)* is called when an intelligent agent wants to add a message to the main blackboard. DEL_BB_STR *(Delete BlackBoard String)* is called when an intelligent agent wants to delete a message from the main blackboard. CHANGE_GROUP_NAME is called when intelligent agent needs to change group. EXIT_CLIPS is called when an intelligent agent is exiting from the system permanently. In the above commands, *Message string* takes the following format :

"<destination> <type> <message>"

The *Destination* field should be the name of an intelligent agent or group currently active in the system, or "ALL" which specifies that the message should be received by all active intelligent agents. *(In the above format, blank characters were used to separate each field.)*

The *Type* field should be "FACT", "RULE" or "COMMAND", which describes the type of the message in the *message string*. If the *type* is FACT, the *message* will be added or deleted, depending on the functions, to shared fact base of the intelligent agent specified by the *destination* field. If the *type* is RULE, the *message* will be added or deleted to the dynamic knowledge base of the intelligent agent(s) as specified in the *destination* field. If the *type* is COMMAND, then *message* will be executed as a command by the intelligent agent specified in the *destination* field. Commands are always removed from the main blackboard after intelligent agents receive a copy of the blackboard.

Message can contain facts, rules or commands. Since the current implementation of the prototype system only uses CLIPS shell to represent an intelligent agent, we have chosen to follow the syntax of the CLIPS to represent facts, rules or commands. If another expert system shell is used to represent an intelligent agent, this common syntax should be observed to transfer facts, rules or commands. It is the intelligent agent's responsibility to translate this common syntax to its own internal syntax.

The following sections describe, in more detail, the process of transferring facts, rules, and commands among intelligent agents.

2.6. Fact Transfer Among Intelligent Agents

An intelligent agent can send fact(s) to an individual intelligent agent, group of intelligent agents or to all intelligent agents in the system. Facts stay in the main blackboard until removed by the sender intelligent agent or by other intelligent agent(s) that has/have the permission to delete the fact. ADD_BB_STR and DEL_BB_STR commands are used to insert, or remove fact(s) from the main blackboard. The following examples show how facts transferred among intelligent agents in the system.

Given the following information :

?y is a string containing "ALL FACT Hi there, I have just joined the system"
?x is a string containing "IA-2 FACT apple is red"
IA-2 is the name of an intelligent agent active in the system.

Then:

(ADD_BB_STR ?y) adds the message *(i.e., the fact)* *"Hi there, I have just joined the system"* to the main blackboard as a fact and all intelligent agents insert this fact to their internal shared fact base.

(DEL_BB_STR ?y) deletes the message *(i.e., the fact)* *"Hi there, I have just joined the system"* from the main blackboard as well as from internal shared fact base of any intelligent agent that has this fact in its shared fact base.

(ADD_BB_STR ?x) adds *"apple is red"* to the main blackboard as a fact and only IA-2 is allowed to insert this fact to its shared fact base.

(DEL_BB_STR ?x) deletes *"apple is red"* from the main blackboard and causes IA-2 to remove this fact from its shared fact base.

2.7. Command Transfer Among Intelligent Agents

An intelligent agent can send command(s) to an individual intelligent agent, group of intelligent agents or to all intelligent agents in the system. Commands do not remain on the main blackboard, the receiver intelligent agent executes the command immediately upon its arrival. Commands are deleted by the receiver intelligent agent(s) as soon as they are executed. ADD_BB_STR function should be used for transferring commands among intelligent agents. DEL_BB_STR function is not available on COMMAND type. Following examples demonstrate how commands are transferred among intelligent agents. In the following examples all the commands are standard commands available in the CLIPS expert system shell.

Given the following information :

?y is a string containing "ALL COMMAND (rules)"
?x is a string containing "IA-2 COMMAND (watch facts)"
?z is a string containing "GROUP1 COMMAND (CHANGE_GROUP_NAME "GROUP2")"
IA-2 is the name of the an intelligent agent active in the system.

Then:

(ADD_BB_STR ?y) all intelligent agents in the system execute *(rules)* command which means print all rules in the intelligent agent knowledge base.

(ADD_BB_STR ?x) IA-2 executes *(watch facts)* command.

(ADD_BB_STR ?z) all intelligent agents in the GROUP1 will change their group name to GROUP2.

All CLIPS commands are supported by the DYNACLIPS. Hence, an intelligent agent can modify the knowledge of other intelligent agents via sending the appropriate command. Application programmer should be careful when designing the system since it is possible to remove static knowledge and local facts of the intelligent agent receiving the commands.

2.8. Rule Transfer Among Intelligent Agent

An intelligent agent can send rule(s) to an individual intelligent agent, group of intelligent agents or to all intelligent agents in the system. Rules stay on the main blackboard until removed by the sender intelligent agent or other intelligent agent(s) that has the right permission to delete the rule. CLIPS format should be followed to represent rules. The type RULE should be used in the type field of the message string. ADD_BB_STR and DEL_BB_STR commands can be used to add or delete rules from the main blackboard. The following presents examples for transferring rule among intelligent agents.

Given the following information :

?x is a string containing "IA-2 RULE *(defrule **rule1** (apple is red) => (facts))* "
?y is a string containing "ALL RULE *(defrule **rule2** (red is color) => (facts))* "

IA-2 is the name of an intelligent agent active in the system.

Then :

(ADD_BB_STR ?x) ***rule1*** will be added to the main blackboard and only IA-2 can insert the ***rule1*** into its dynamic knowledge base.

(ADD_BB_STR ?y) ***rule2*** will be added to the main blackboard and all intelligent agents can insert ***rule2*** into their dynamic knowledge base.

2.9. Knowledge Transfer Among Intelligent Agents

Knowledge can be exchanged among intelligent agents by using combination of facts, rules and commands transfers. Different methodologies can be used for knowledge transfer; knowledge can be exchanged among intelligent agents in temporary or permanent bases.

Under temporary knowledge transfer option, the sender intelligent agent specifies the rule(s) that needs to be transferred as well as specifying when the rules needs to be removed from dynamic knowledge base of the receiver intelligent agent. The following example shows how to transfer a temporary knowledge among intelligent agents.

Given the following information :

?x is a string that contains the following :

" ALL COMMAND (defrule rule1 (lights are on) =>

(turn of the lights)

(assert (lights are off)) "

?y is a string that contains the following :

" ALL COMMAND (defrule rule2 (lights are off) =>

(undefrule rule1)

(undefrule rule2)) "

Then:

(ADD_BB_STR ?x)

(ADD_BB_STR ?y)

In the above example, two rules were broadcasted to all intelligent agents in the system. All intelligent agents will insert these two rules into their dynamic knowledge base. The rules will remain in the intelligent agent's dynamic knowledge base until *rule1* is fired. *Rule2* will be fired after *rule1*, which would cause *rule1* and itself to be removed from the intelligent agents' dynamic knowledge bases. Using type COMMAND will cause that the two rules be deleted from the main blackboard as soon as all intelligent agents have read them into their dynamic knowledge base. By eliminating the second function *(i.e., (ADD_BB_STR ?y))* from the previous example, the rule can be placed permanently in the receiver intelligent agents' knowledge bases. Hence, the knowledge encoded in the rule can be used by the receiver intelligent agent from that point on. The following example demonstrates this concept.

Given the following information

?x is a string that contains the following :

"ALL COMMAND (defrule rule1 (lights are on) =>

(turn of the lights)

(assert (lights are off)) "

Then:

(ADD_BB_STR ?x)

Type RULE should be used to transfer knowledge when the intelligent agents are joining, exiting or re-joining the framework continuously. In this case, knowledge stays in the main blackboard until deleted explicitly by the sender intelligent agent.

3. CONCLUSIONS AND FURTHER STUDIES

By introducing simple communication protocols among intelligent agents, we have introduced a framework through which intelligent agents can exchange knowledge in a dynamic environment. Using the DYNACLIPS common knowledge can be maintained by one intelligent agent and broadcasted to the other intelligent agents when necessary.

In a dynamic environment, intelligent agents must be responsive to unanticipated conditions. When such conditions occur, an intelligent agent may be required to terminate previously planned and scheduled courses of action, and replan, reschedule, start new activities, and initiate a new problem solving process, in order to successfully respond to the new conditions. Problems occur when an intelligent agent does not have sufficient knowledge to properly respond to the new condition. In order to successfully respond to unexpected events in dynamic environments, it is imperative to have the capability of dynamic knowledge exchange among intelligent agents.

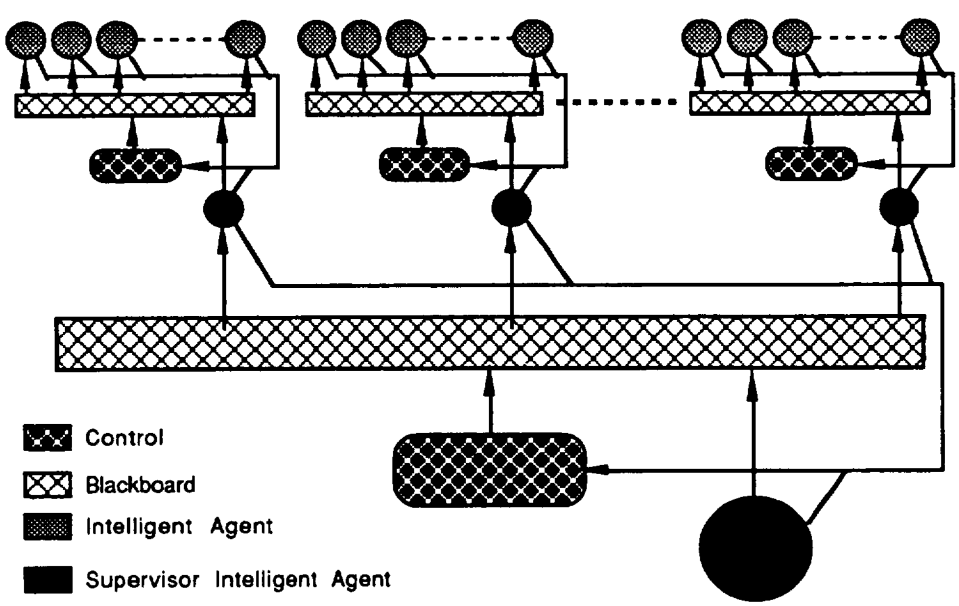

Figure 6. An Example of a Framework which Includes Multilayer Aspects

We believe that dynamic knowledge exchange would be an important feature for any application in which unanticipated conditions or events occur. Using the proposed dynamic knowledge exchange capability, cooperative problem solving sessions can be initiated where each intelligent agent can share its problem relevant knowledge with other intelligent agents to resolve the problem. An obvious advantage of this capability is the elimination of redundant knowledge and hence the improved utilization of the system memory capacity. In addition, by using this framework a form of learning can take place and thus additional problem solving knowledge is created.

The basic framework presented in this research could be extended to include a multilayer environment. This can be done by providing supervisory blackboard and control components to create a single entity by combining separate frameworks. (See Figure 6.)

The proposed framework can easily be expanded to accommodate more than one blackboard when it is necessary. In this case one control module is associated with each blackboard as in Figure 7. The basic process of communication with the control modules would be the same as presented in the previous sections.

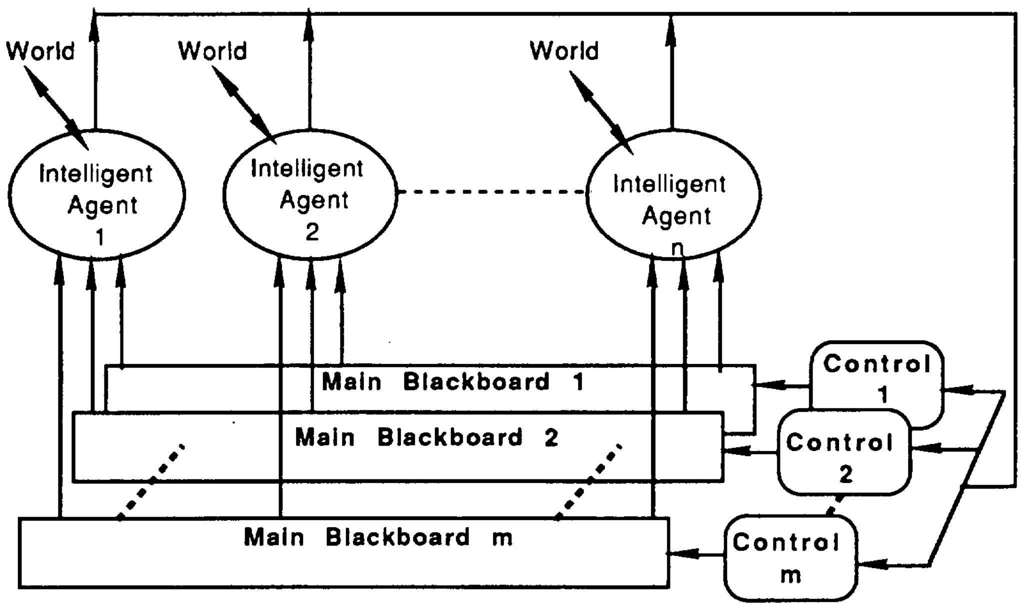

Figure 7. Using Multiple Blackboard

The prototype system is currently running on a single computer system using multiprocessing facilities. By using networking, it is possible to extend the system functionality to support distributed environments. Hence, intelligent agents can be geographically distributed but able to communicate via the system main blackboard.

Since the current implementation of the DYNACLIPS only uses CLIPS shell to represent an intelligent agent, we have chosen to follow the syntax of the CLIPS to represent facts, rules or commands. Knowledge Interface Format (KIF) [15,16] can also be used in the future to transfer facts, rules or commands.

REFERENCES

[1] Gilmore, J. F., Roth, S. P. and Tynor S. D. *A Blackboard System for Distributed Problem Solving.* Blackboard Architectures and Applications. Academic Press, San Diego, CA, 1989.

[2] Nii, H. P. *Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures* , AI Magazine, Summer 1986, pp. 38-53.

[3] Nii, H. P. *Blackboard Systems: Blackboard Application Systems, Blackboard Systems from a Knowledge Engineering Perspective.* AI Magazine, August 1986, pp. 82-106.

[4] Diaz, A. C. and Orchard, R. A. *A Prototype Blackboard Shell using CLIPS.* Fifth International Conference on AI in Engineering, Boston, MA, July 1990.

[5] Orchard, R. A. and Diaz, A. C. *BB_CLIPS: Blackboard Extensions to CLIPS,* Proceeding of the First CLIPS User's Group Conference, Houston, TX, 1990.

[6] Dodhiawala, R. T., Sridharan, N. S. and Pickering C. *A Real-Time Blackboard Architecture,* Blackboard Architectures and Applications. Academic Press, San Diego, CA, 1989.

[7] Golstein, G. *PRAIS: Distributed, Real-Time Knowledge_Based Systems Made Eassy.* Proceeding of the First CLIPS User's Group Conference, Houston, TX, 1990.

[8] Myers, L. Johnson, C and Johnson, D. *MARBLE: A Systems for Executing Expert System in Parallel.* Proceeding of the First CLIPS User's Group Conference, Houston, TX, 1990.

[9] Schultz, R. D. and Stobie, I. C. *Building Distributed Rule-Based Systems Using the AI Bus.* Proceeding of the First CLIPS User's Group Conference, Houston, TX, 1990.

[10] Gallagher K. Q. and Corkill, D. D. *Performance Aspects of GBB,* Blackboard Architectures and Applications. Academic Press, San Diego, CA, 1989.

[11] Blackboard Technology Group, inc. *The Blackboard Problem Solving Approach,* AI Review, Summer 1991, pp. 37-32

[12] Gilmore, J. F., Roth, S. P. and Tynor S. D. *A Blackboard System for Distributed Problem Solving.* Blackboard Architectures and Applications. Academic Press, San Diego, CA, 1989.

[13] Rice, J., Aiello, N., and Nii, H. P. *See How They Run... The Architecture and Performance of Two Concurrent Blackboard Systems.* Blackboard Systems. Addison Wesley, Reading, Mass, 1988.

[14] Erman, L. D., Hayes-Roth, F., Lesser, R. V. and Reddy, D. R. *The Hearsay-II Speech-Understanding System : Integrating Knowledge to Resolve Uncertanity.* Blackboard Systems. Addison Wesley, Reading, Mass, 1988.

[15] Genesereth, M.R., Fikes, R. E. *Knowledge Interchange Format Reference Menual,* Stanford University Logic Group, 1992.

[16] Genesereth, M.R., Ketchpel, S. P., *Software Agents,* ACM Communication, July 1994, pp. 48-53.

[17] Cengeloglu, Y., Sidani, T. and Sidani, A. *Inter/Intra Communication in Intelligent Simulation and Training Systems (ISTS).* 14th Conference on Computers and Industrial Engineering, Cocoa Beach,March 1992.

[18] CLIPS Version 5.1 User's Guide, NASA Lyndon B. Johnson Space Center, Software Technology Branch, Houston, TX, 1991